Cray Scientific Libraries : Usage, hybrid modes, advanced performance

Jeff Larkin

larkin@cray.com

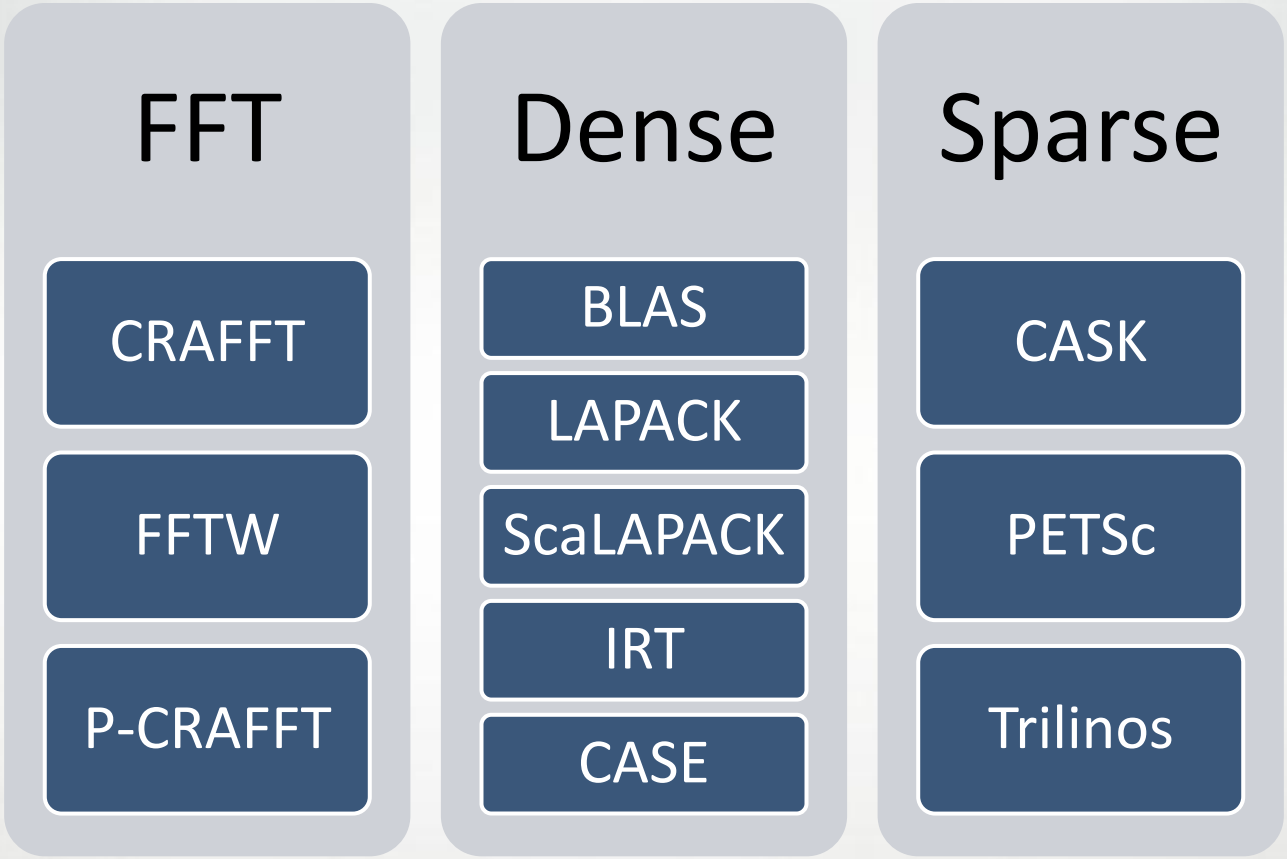
Cray Scientific Libraries Today

- Goal of scientific libraries

  Improve Productivity at optimal performance

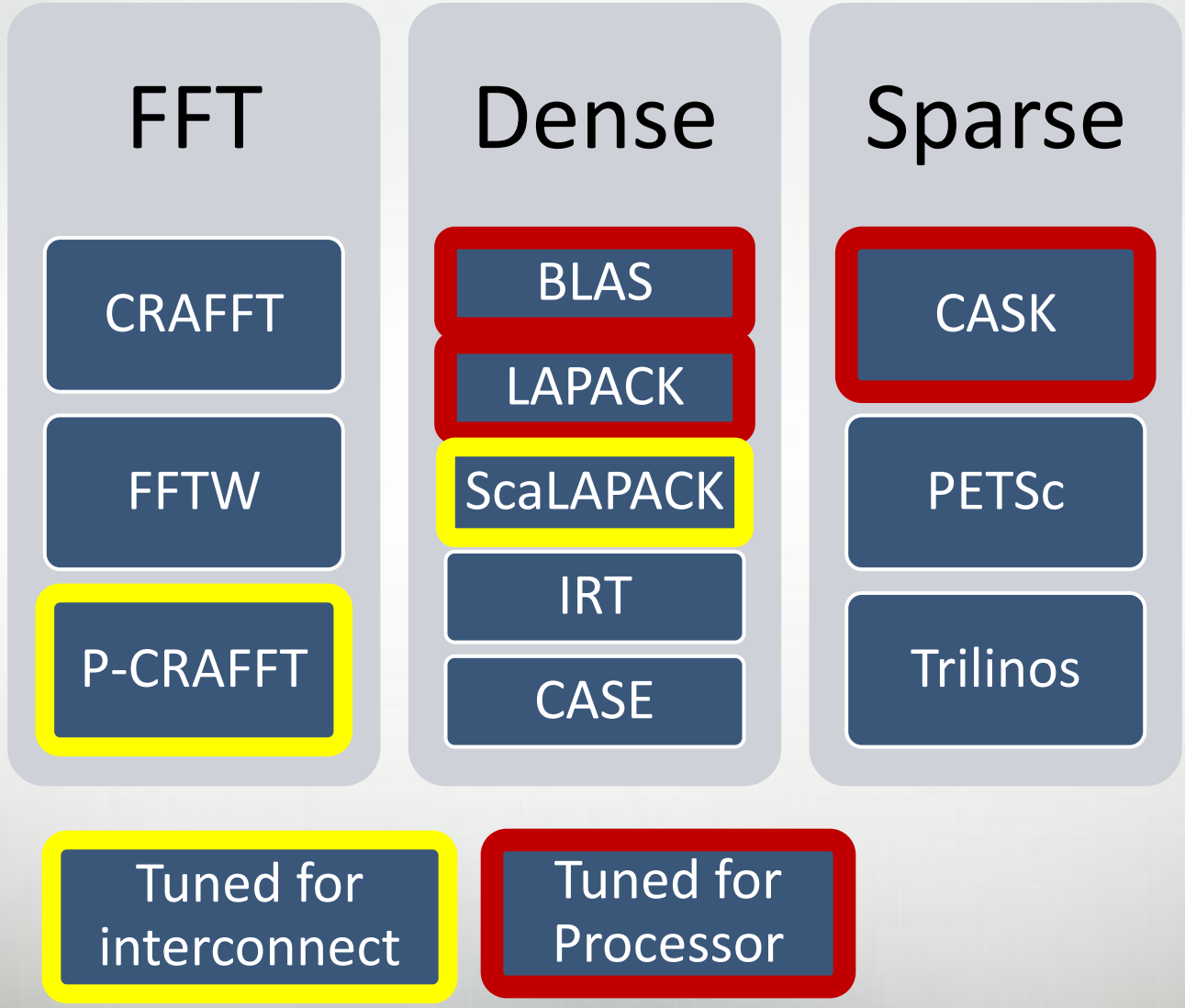
- Cray use four concentrations to achieve this
  - **Standardization**
    - Use standard or “de facto” standard interfaces whenever available
  - **Hand tuning**
    - Use extensive knowledge of target processor and network to optimize common code patterns
  - **Auto-tuning**
    - Automate code generation and a huge number of empirical performance evaluations to configure software to the target platforms
  - **Adaptive Libraries**
    - Make runtime decisions to choose the best kernel/library/routine

Cray Scientific Libraries

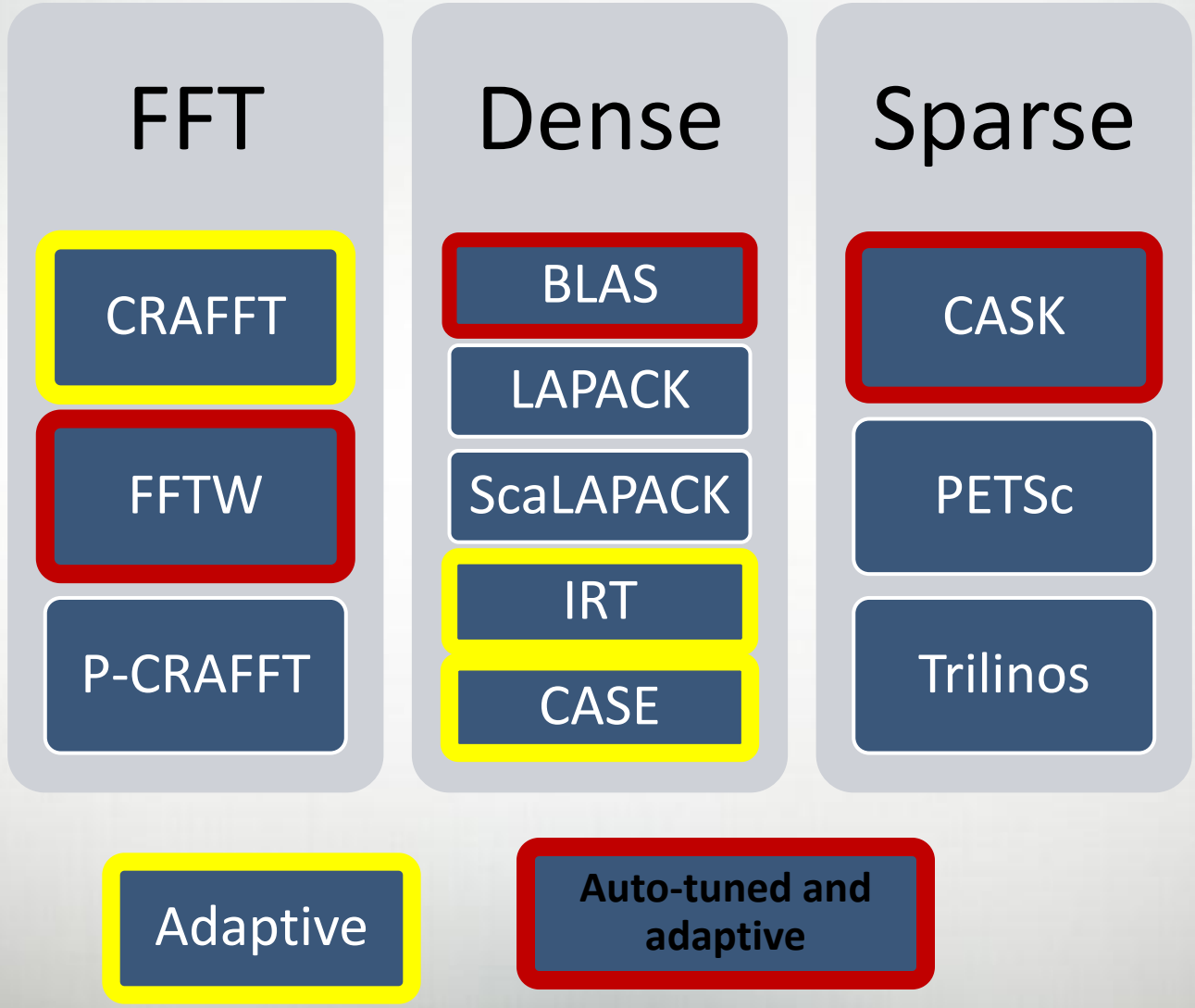


IRT – Iterative Refinement Toolkit
CASK – Cray Adaptive Sparse Kernels
CRAFFT – Cray Adaptive FFT
CASE – Cray Adaptive Simplified Eigensolver

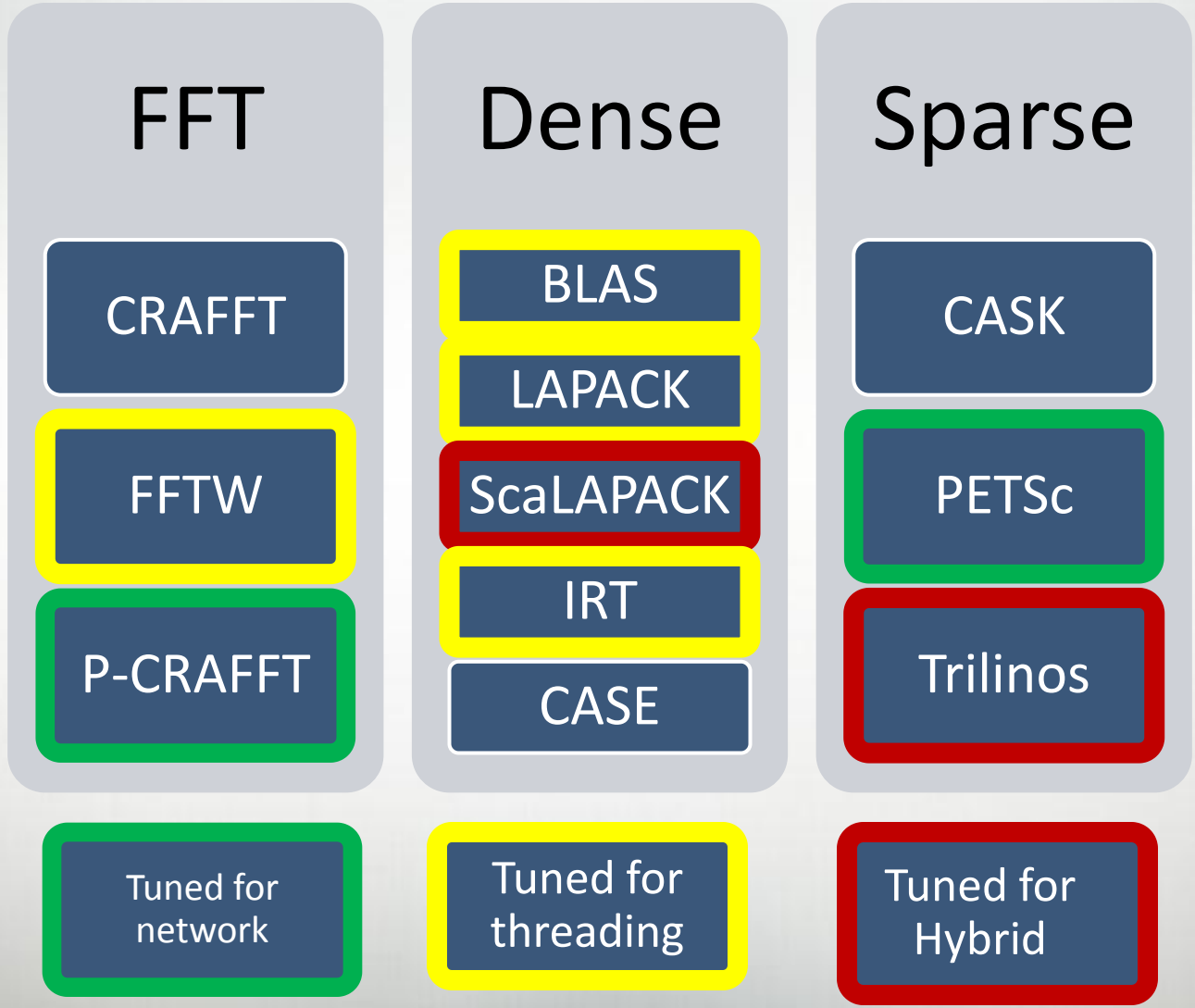
Cray Scientific Libraries - Tunings



Cray Scientific Libraries – autotuning focus



Cray Scientific Libraries – tuning focus



LibSci Provides…

- BLAS
- LAPACK
- SCALAPACK
- BLACS
- PBLAS
- ACML
- FFTW 2&3
- PETSC
- TRILINOS
- IRT*
- MUMPS
- ParMetis
- SuperLU
- SuperLU_dist
- Hypre
- Scotch
- Sundials
- CASK*
- CRAFFT*
- CASE*

* Cray-specific

General usage information

- There are many libsci libraries on the systems
- One for each of
  - Compiler (intel, cray, gnu, pathscale, pgi)
  - Single thread, multiple thread
  - Target (istanbul)
- Best way to use libsci is to ignore all of this
- Load the xtpe-module (loaded here by default)
  - module load xtpe-istanbul
- Cray's compiler drivers will link the library automatically
- PETSc, Trilinos, fftw, acml all have their own module

Adding another library

- Perhaps you want to link another library such as ACML
- This can be done. If the library is provided by Cray, then load the module. The link will be performed with the libraries in the correct order.
- If the library is not provided by Cray and has no module, add it to the link line.
  - Items you add to the explicit link will be in the correct place
- Note, to get explicit BLAS from ACML but scalapack from libsci
  - Load acml module. Explicit calls to BLAS in code resolve from ACML
  - BLAS calls from the scalapack code will be resolved from libsci (no way around this)

Making sure you have the right library

- I recommend adding options to the linker to make sure you have the correct library loaded.

- –Wl, -ydgemm_ will return :

  cc -L./ -o mmulator blas_test.o netlib_dgemm.o -Wl,-ydgemm_

  blas_test.o: reference to dgemm_

  /opt/xt-libsci/10.4.9/cray/lib/libsci.a(dgemm.o): definition of dgemm_

OpenMP BLAS

- Threading capabilities in previous libsci versions were poor
  - Used PTHREADS (more explicit affinity etc)
  - Required explicit linking to a _mp version of libsci
  - Was a source of concern for some applications that need hybrid performance and interoperability with openMP
- LibSci 10.4.2 February 2010
  - OpenMP-aware LibSci
  - Allows calling of BLAS inside or outside parallel region
  - Single library supported (there is still a single thread lib)
- Usage – load the xtpe module for your system (istanbul)

GOTO_NUM_THREADS outmoded – use OMP_NUM_THREADS

OpenMP LibSci

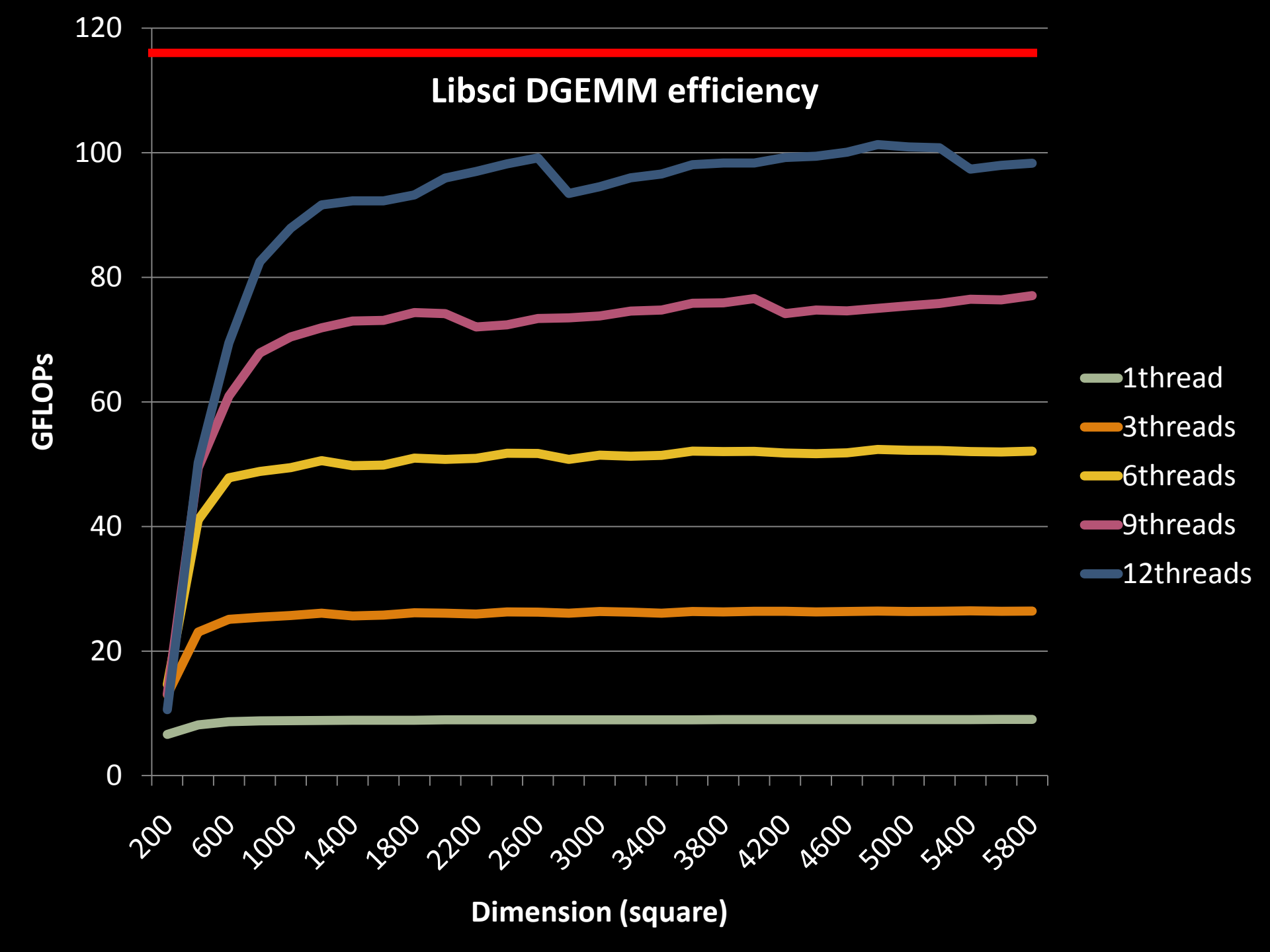
- Allows seamless calling of the BLAS within or without a parallel region

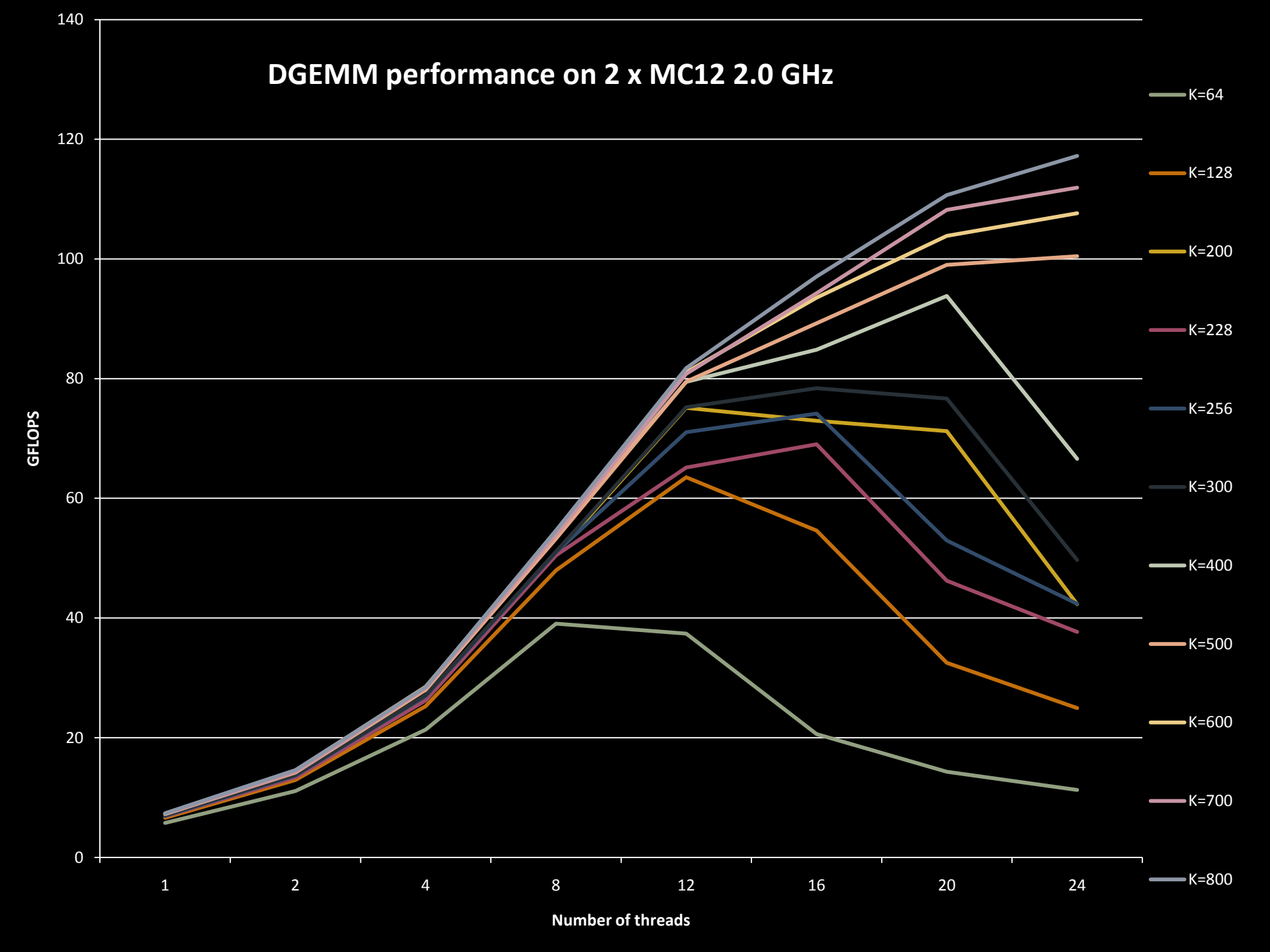
e.g. OMP_NUM_THREADS = 6

```
call dgemm(…)   threaded dgemm is used with 6 threads
!$OMP PARALLEL DO
do
   call dgemm(…)   single thread dgemm is used
end do
```

Other situations

- OMP_NUM_THREADS controls both types of parallelism
- Library sets buffers based on OMP_NUM_THREADS on first call
- The side effect to this model it is not possible to have ‘split-parallelism’
- Changing dynamically OMP_SET_NUM_THREADS is not possible!
- We are working on a more flexible scheme for release early 2011


Libsci DGEMM efficiency
GFLOPs
120
100
80
60
40
20
0
200
600
1000
1400
1800
2200
2600
3000
3400
3800
4200
4600
5000
5400
5800
Dimension (square)
1thread
3threads
6threads
9threads
12threads


DGEMM performance on 2 x MC12 2.0 GHz
GFLOPS
Number of threads
0
20
40
60
80
100
120
140
1
2
4
8
12
16
20
24
K=64
K=128
K=200
K=228
K=256
K=300
K=400
K=500
K=600
K=700
K=800

BLAS2 and BLAS1 performance

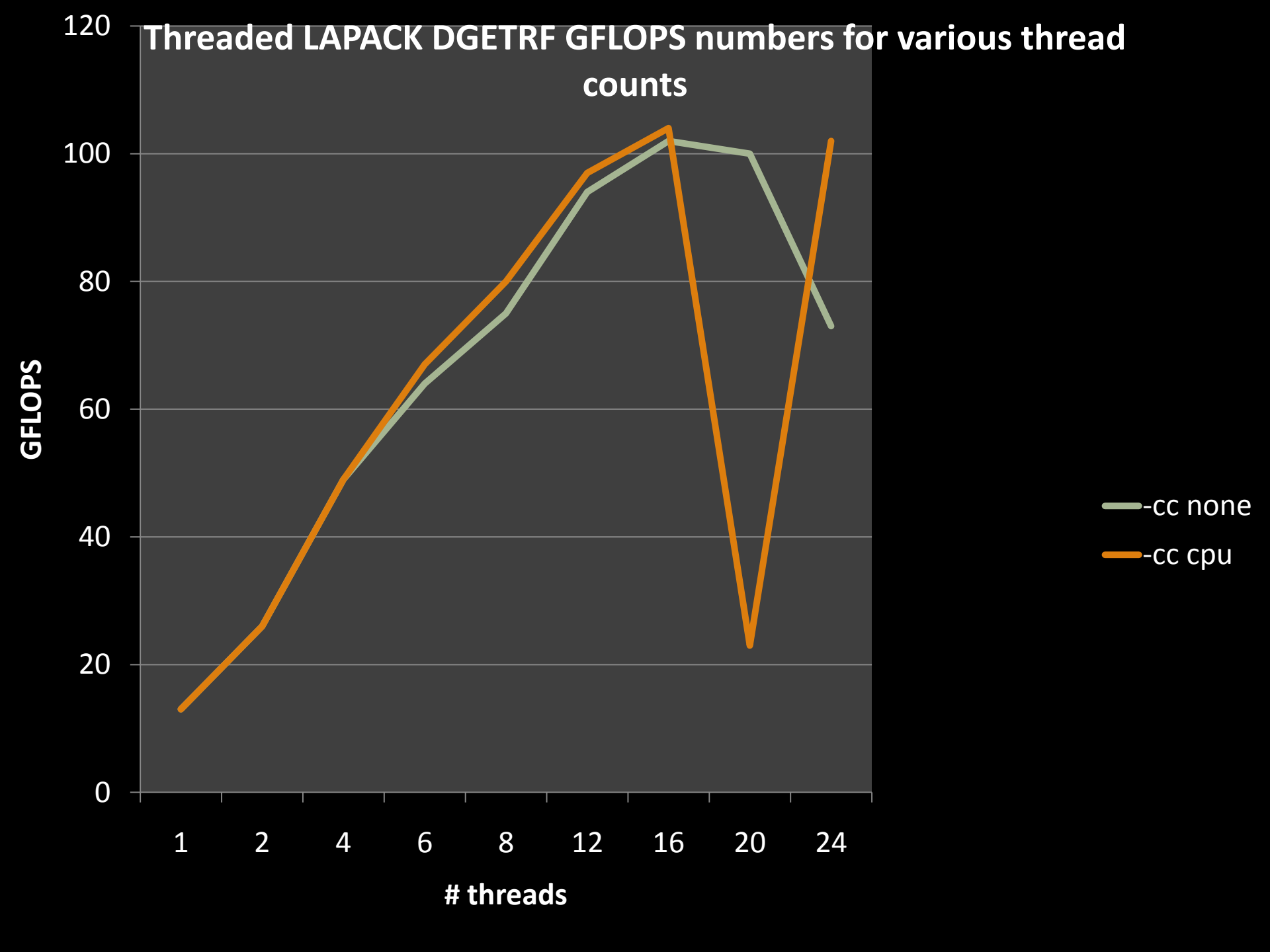
- Memory-bound code doesn't thread well.
- But, you can still obtain a little speed-up because you use more memory channels when you use threads.
- Some of the BLAS2 can exhibit some speed-up with threading

Usage

- module load xtpe-instanbul
- No need to explicit link
- Add –Wl,-ydgemm_ to link line
- Set OMP_NUM_THREADS in job script
- Run with aprun –n 1 –d6 ./exec (for 6 threads)

Threaded LAPACK

- LAPACK is the very popular linear algebra library for on-node
- Cray's implementation of LAPACK is tuned.
- LAPACK is threaded in the same way as BLAS
- In some routines, the threading is at a higher level than the BLAS updates (LU, Cholesky, QR, some eigensolvers)
- Usage is exactly the same as with the BLAS

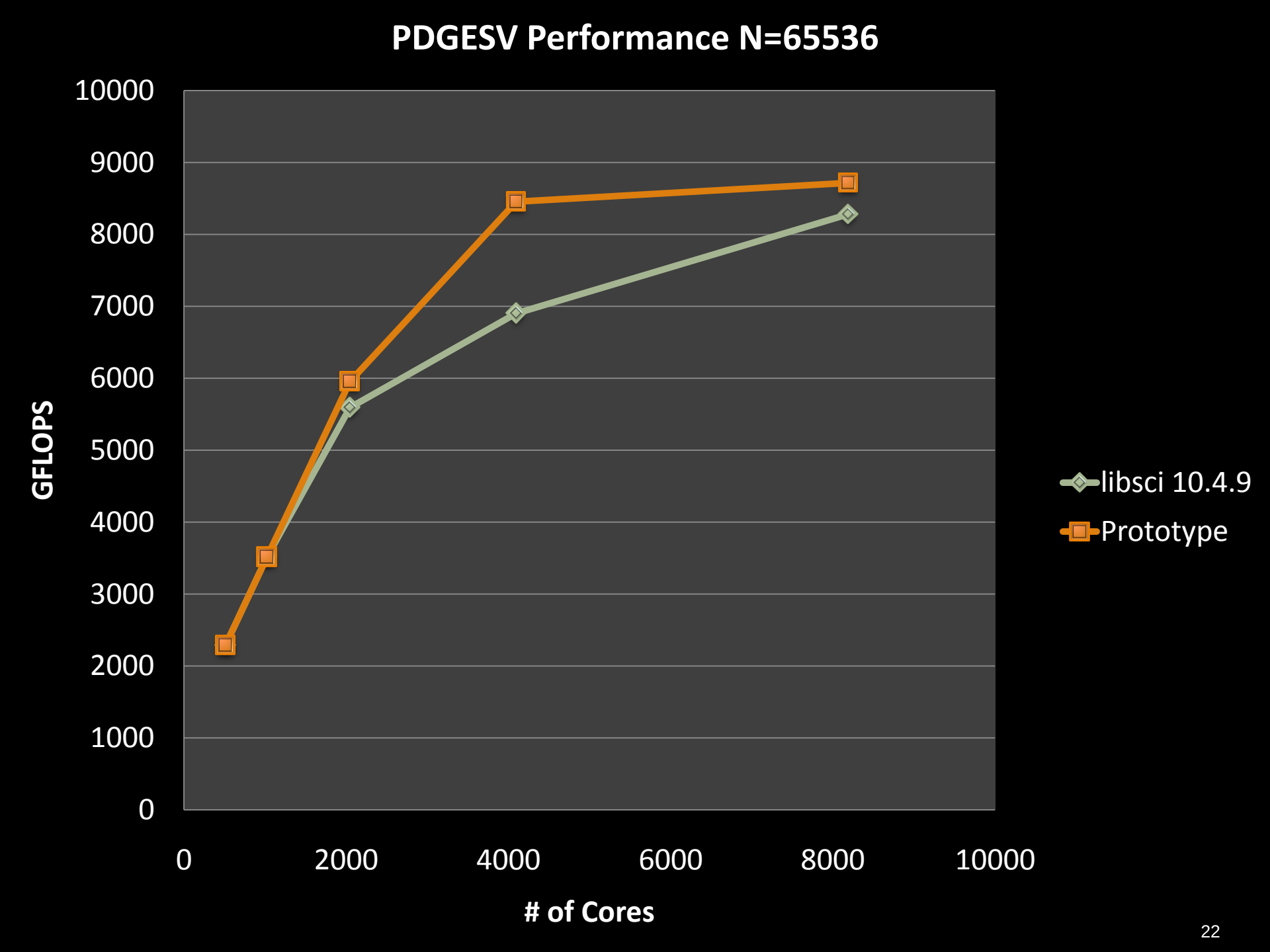

Threaded LAPACK DGETRF GFLOPS numbers for various thread counts
GFLOPS
120
100
80
60
40
20
0
1
2
4
6
8
12
16
20
24
threads
-cc none
-cc cpu

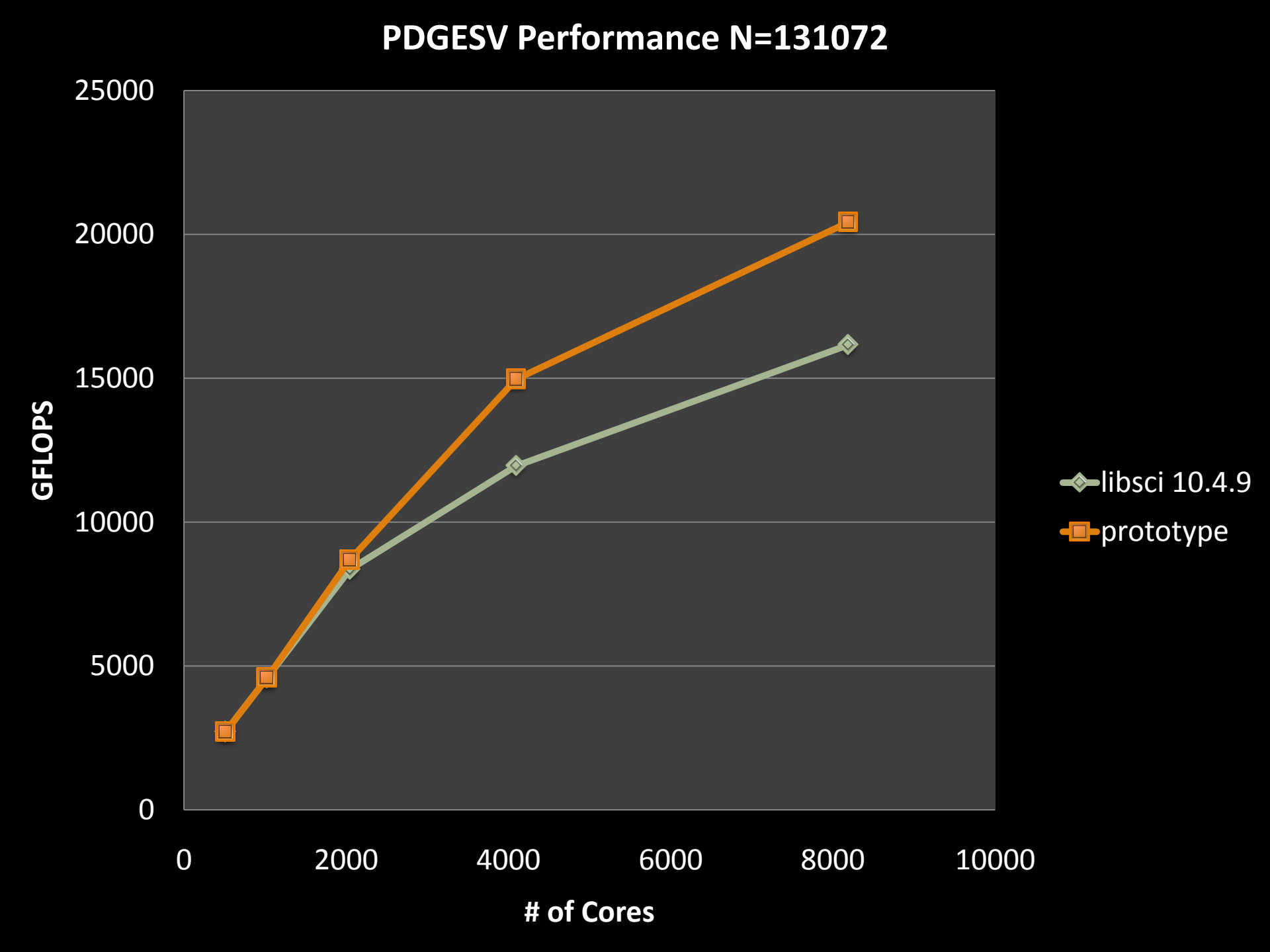
ScaLAPACK

- ScaLAPACK is the near-standard parallel linear algebra library
- Uses distributed memory BLAS, PBLAS
- Uses BLACS for communication
- Using scalapack across nodes and threaded BLAS within nodes is the simplest way to obtain hybrid MPP + thread functionality
- Cray have tuned ScaLAPACK on previous machines, and we are doing so now on XE6.

Tuning ScaLAPACK for XE6

- Cray has a strong track record of tuning parallel linear algebra for older systems – T3E, X1
- Used shmem to replace key communication schemes
- On XE, use many of the same techniques and some new ones
  - Focusing on the LU, Cholesky, divide and conquer eigensolver, tridiagonal reduction
  - Using more asynchronous communications in factorizations
  - Replacing MPI with co-array fortran and shmem


PDGESV Performance N=65536
GFLOPS
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
0
2000
4000
6000
8000
10000
# of Cores
libsci 10.4.9
Prototype


PDGESV Performance N=131072
25000
20000
15000
10000
5000
0
GFLOPS
0
2000
4000
6000
8000
10000
# of Cores
libsci 10.4.9
prototype

Using ScaLAPACK in hybrid mode on XE6

- Use the number of scalapack grid points you want to correspond to the number of MPI ranks you want
- Rely on the BLAS to operate with the number of threads you desire
- Use OMP_NUM_THREADS and the aprun options to set the number of threads you need for on-node parallelism
- Set the threads per node from libsci BLAS with OMP_NUM_THREADS
- Use aprun options –n and –d for nodes and threads

Iterative Refinement Toolkit

- Mixed precision can yield a big win on x86 machines.
- SSE (and AVX) units issue double the number of single precision operations per cycle.
- On CPU, single precision is always 2x as fast as double
- Accelerators sometimes have a bigger ratio
  - Cell – 10x
  - Older NVIDIA cards – 7x
  - New NVIDIA cards (2x)
  - Newer AMD cards (> 2x)
- IRT is a suite of tools to help exploit single precision
  - A library for direct solvers
  - An automatic framework to use mixed precision under the
  - A domain-specific language and preprocessor to convert codes to use mixed precision without active code change

Iterative Refinement Toolkit - Library

- Various tools for solves linear systems in mixed precision
- Obtaining solutions accurate to double precision
  - For well conditioned problems
- Serial and Parallel versions of LU, Cholesky, and QR
- 2 usage methods
  - **IRT Benchmark routines**
    - Uses IRT 'under-the-covers' without changing your code
      - Simply set an environment variable
      - Useful when you cannot alter source code
  - **Advanced IRT API**
    - If greater control of the iterative refinement process is required
      - Allows
        - condition number estimation
        - error bounds return
        - minimization of either forward or backward error
        - 'fall back' to full precision if the condition number is too high
        - max number of iterations can be altered by users

IRT library usage

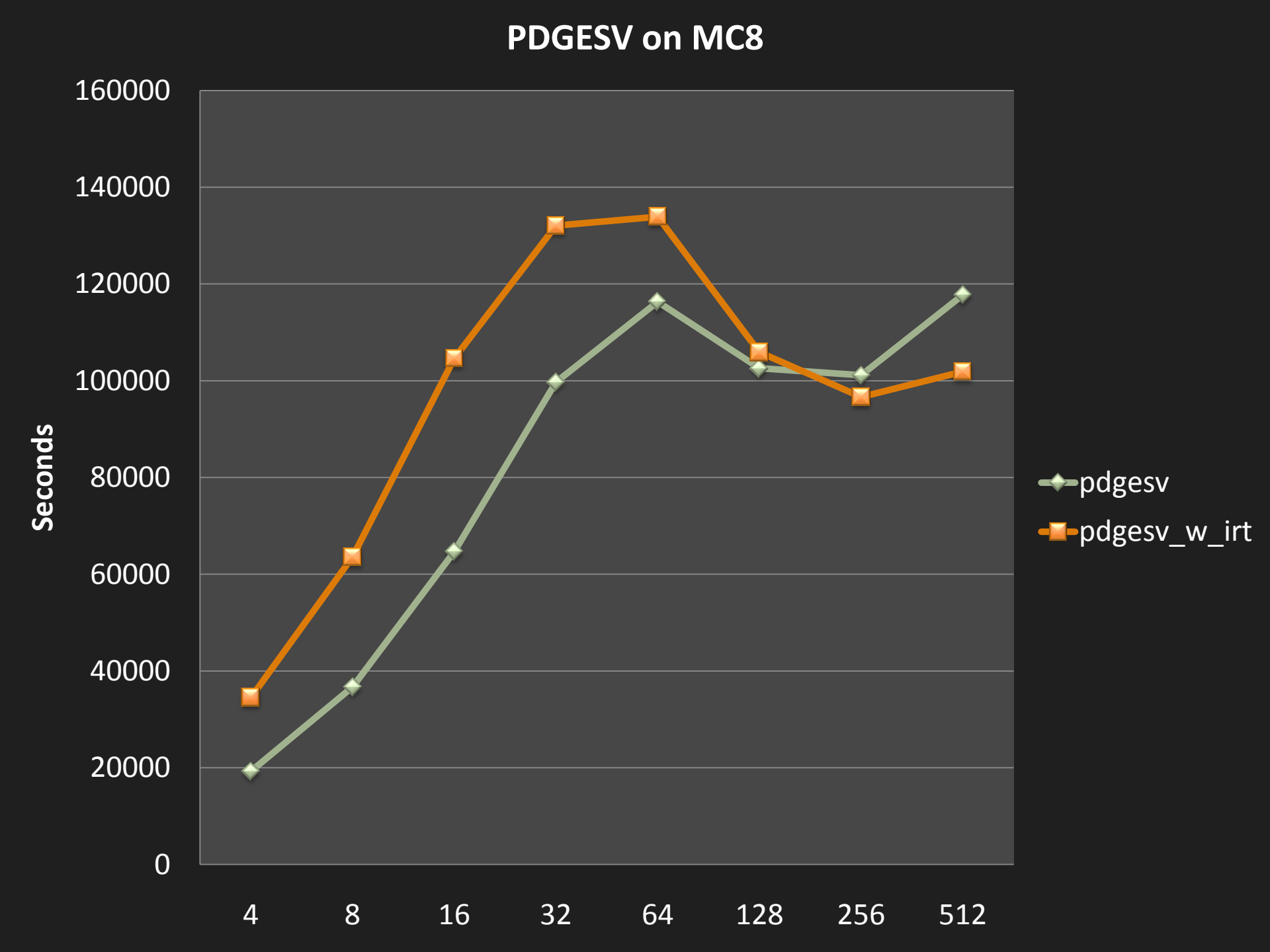
Decide if you want to use advanced API or benchmark API

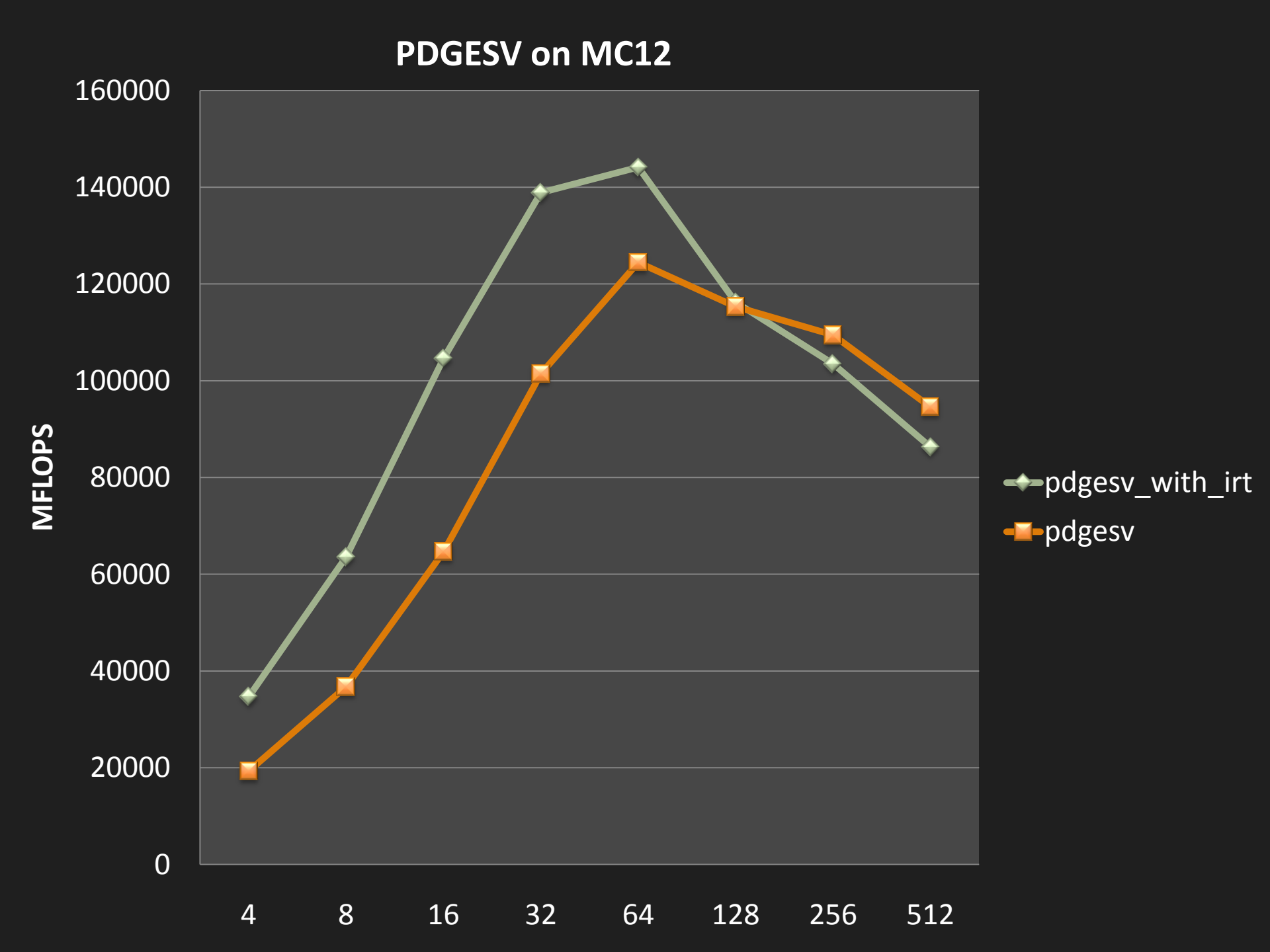
benchmark API :

setenv IRT_USE_SOLVERS 1

advanced API :

1. locate the factor and solve in your code (LAPACK or ScaLAPACK)
2. Replace factor and solve with a call to IRT routine
   - e.g. dgesv -> irt_lu_real_serial
   - e.g. pzgesv -> irt_lu_complex_parallel
   - e.g pzposv -> irt_po_complex_parallel
3. Set advanced arguments
   - Forward error convergence for most accurate solution
   - Condition number estimate
   - “fall-back” to full precision if condition number too high


PDGESV on MC8
160000
140000
120000
100000
80000
60000
40000
20000
0
Seconds
4
8
16
32
64
128
256
512
pdgesv
pdgesv_w_irt


PDGESV on MC12
160000
140000
120000
100000
80000
60000
40000
20000
0
MFLOPS
4
8
16
32
64
128
256
512
pdgesv_with_irt
pdgesv

Cray Adaptive FFT (CRAFFT)

- Serial CRAFFT is largely a productivity enhancer
- Some FFT developers have problems such as
  - Which library choice to use?
  - How to use complicated interfaces (e.g., FFTW)
- Standard FFT practice
  - Do a plan stage
  - Do an execute
- CRAFFT is designed with simple-to-use interfaces
  - Planning and execution stage can be combined into one function call
  - Underneath the interfaces, CRAFFT calls the appropriate FFT kernel

CRAFFT usage

1. Load module fftw/3.2.0 or higher.
2. Add a Fortran statement “use crafft”
3. call crafft_init()
4. Call crafft transform using none, some or all optional arguments (as shown in red)

In-place, implicit memory management :

```
call crafft_z2z3d(n1,n2,n3,input,ld_in,ld_in2,isign)
```

in-place, explicit memory management

```
call crafft_z2z3d(n1,n2,n3,input,ld_in,ld_in2,isign,work)
```

out-of-place, explicit memory management :

```
crafft_z2z3d(n1,n2,n3,input,ld_in,ld_in2,output,ld_out,ld_out2,isign,work)
```

Note : the user can also control the planning strategy of CRAFFT using the CRAFFT_PLANNING environment variable and the do_exe optional argument, please see the intro_crafft man page.

Parallel CRAFFT

- Parallel CRAFFT is meant as a performance improvement to FFTW2 distributed transforms
  - Uses FFTW3 for the serial transform
  - Uses ALLTOALL instead of ALLTOALLV where possible
  - Overlaps the local transpose with the parallel communications
  - Uses a more adaptive communication scheme based on input
- Can provide impressive performance improvements over FFTW2
- Currently implemented
  - complex-complex
  - Real-complex and complex-real
  - 3-d and 2-d
  - In-place and out-of-place
  - 1 data distribution scheme but looking to support more (please tell us)
  - C language support for serial and parallel
  - Generic interfaces for C users (use C++ compiler to get these)

parallel CRAFFT usage

1. Add “use crafft” to Fortran code
2. Initialize CRAFFT using crafft_init
3. Assume MPI initialized and data distributed (see manpage)
4. Call crafft, e.g. (optional arguments in red)

2-d complex-complex, in-place, internal mem management :

```
call crafft_pz2z2d(n1,n2,input,isign,flag,comm)
```

2-d complex-complex, in-place with no internal memory :

```
call crafft_pz2z2d(n1,n2,input,isign,flag,comm,work)
```

2-d complex-complex, out-of-place, internal mem manager :

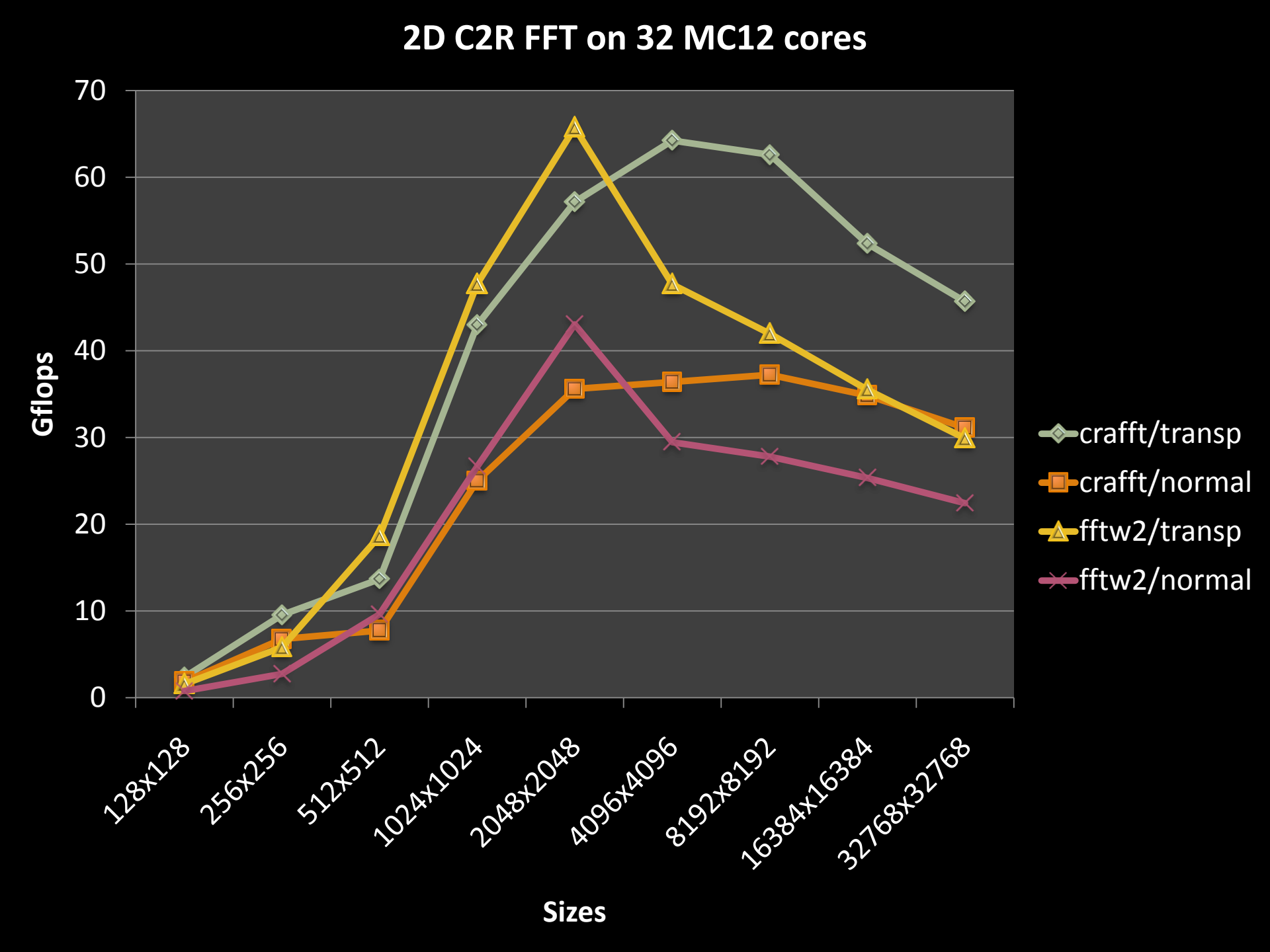
```
call crafft_pz2z2d(n1,n2,input,output,isign,flag,comm)
```

2-d complex-complex, out-of-place, no internal memory :

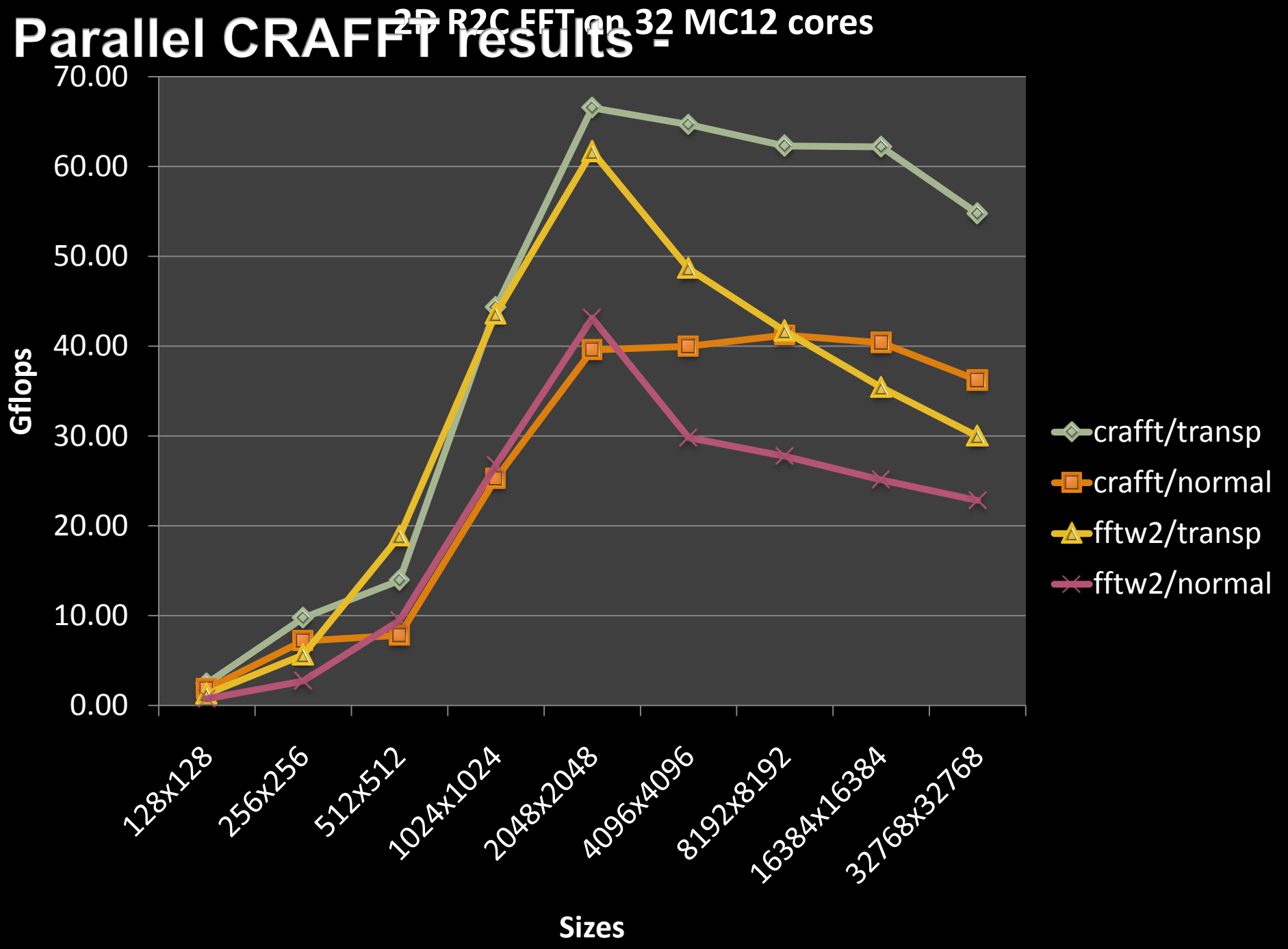
```
crafft_pz2z2d(n1,n2,input,output,isign,flag,comm,work)
```

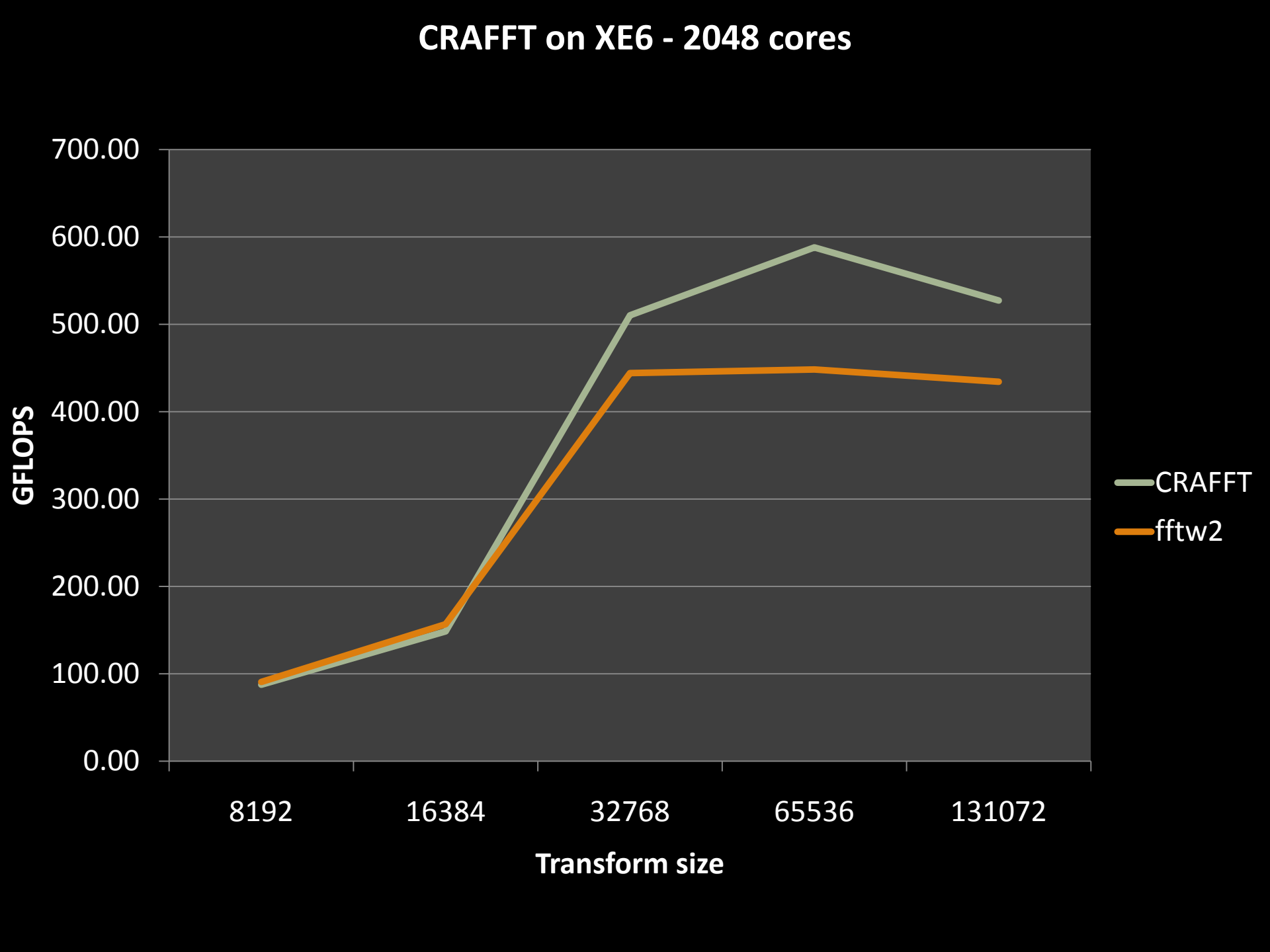
Each routine above has manpage. Also see 3d equivalent :

```
man crafft_pz2z3d
```


2D C2R FFT on 32 MC12 cores
Gflops
70
60
50
40
30
20
10
0
128x128
256x256
512x512
1024x1024
2048x2048
4096x4096
8192x8192
16384x16384
32768x32768
Sizes
crafft/transp
crafft/normal
fftw2/transp
fftw2/normal

Parallel CRAFFT results -




CRAFFT on XE6 - 2048 cores
700.00
600.00
500.00
400.00
300.00
200.00
100.00
0.00
GFLOPS
8192
16384
32768
65536
131072
Transform size
CRAFFT
fftw2

Sparse

- At one time Cray provided both
  - Custom sparse direct solvers
  - Custom sparse iterative solvers
- There has been an evolution towards using standardized frameworks such as Trilinos & PETSc
- Today, we attempt to provide that same performance boost while maintaining productivity
- CASK library – optimizes sparse matrix operations on Cray computers whilst being invisible to the user
  - Cray Trilinos distribution
  - Cray PETSc distribution

PETSc (Portable, Extensible Toolkit for Scientific Computation)

- Serial and Parallel versions of sparse iterative linear solvers
  - Suites of iterative solvers
    - CG, GMRES, BiCG, QMR, etc.
  - Suites of preconditioning methods
    - IC, ILU, diagonal block (ILU/IC), Additive Schwartz, Jacobi, SOR
  - Support block sparse matrix data format for better performance
  - Interface to external packages (ScaLAPACK, SuperLU_DIST)
  - Fortran and C support
  - Newton-type nonlinear solvers
- Extremely large user community in US and Europe
- http://www-unix.mcs.anl.gov/petsc/petsc-as

Usage and External Packages

- Cray provides
  - Hypre: scalable parallel preconditioners
  - ParMetis: parallel graph partitioning package
  - MUMPS: parallel multifrontal sparse direct solver
  - SuperLU: sequential version of SuperLU_DIST
- To use Cray-PETSc, load the appropriate module :

  module load petsc

  (or) module load petsc-complex

  (no need to load a compiler specific module)
- Treat the Cray distribution as your local PETSc installation

  PETSc is not threaded!

Trilinos

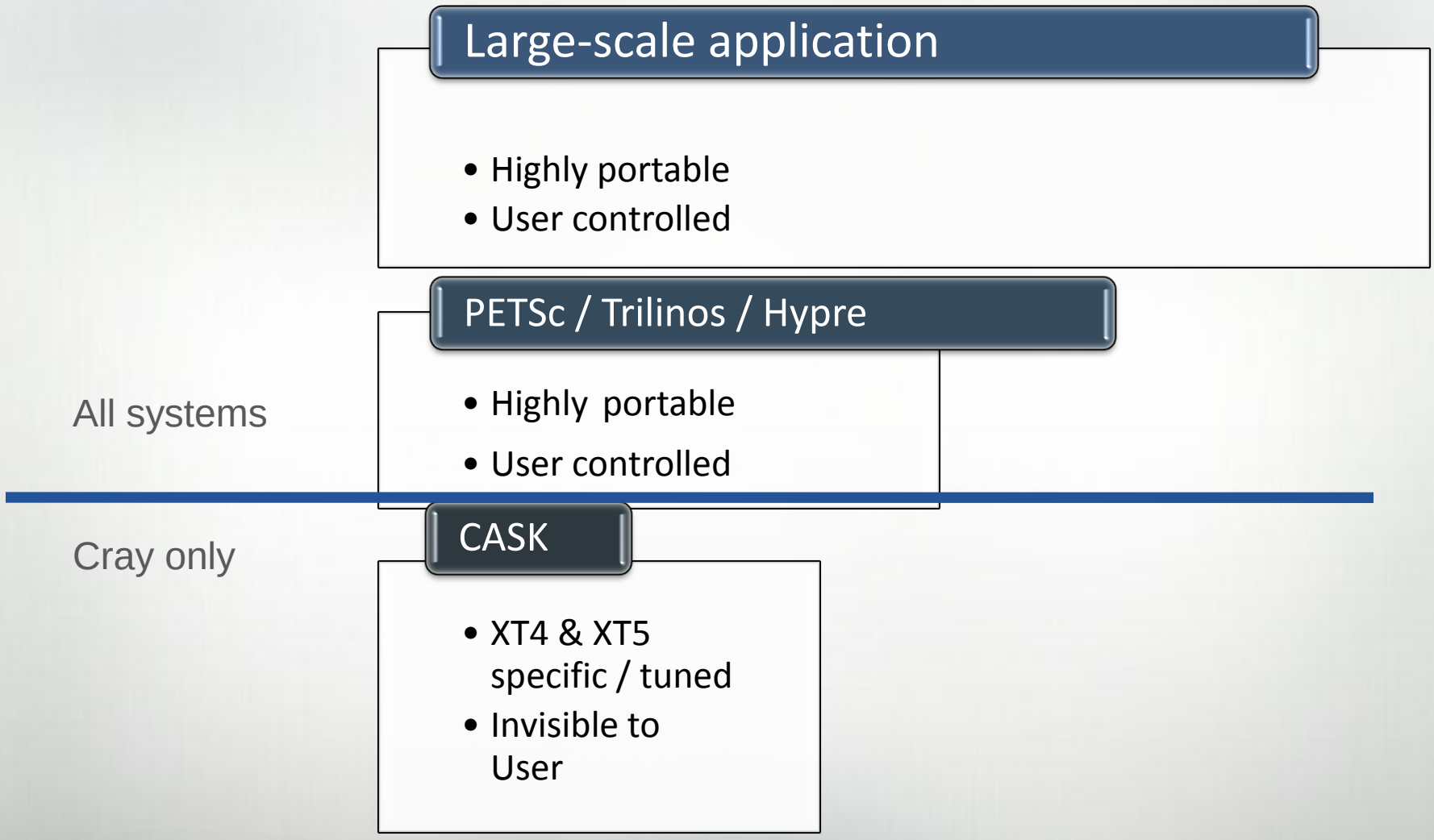
- The Trilinos Project http://trilinos.sandia.gov/

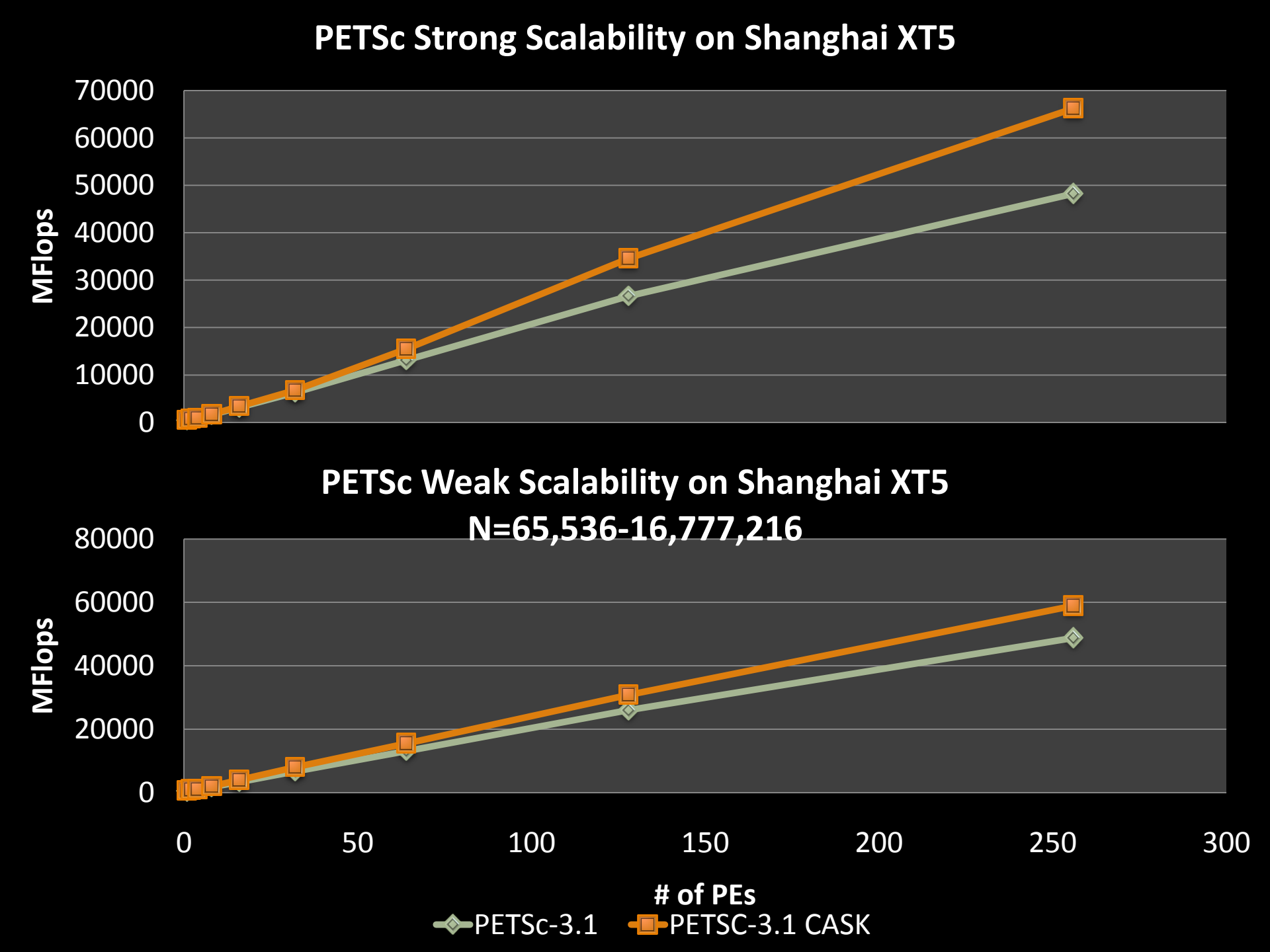
  “an effort to develop algorithms and enabling technologies within an object-oriented software framework for the solution of large-scale, complex multi-physics engineering and scientific problems”
- A unique design feature of Trilinos is its focus on packages.
- Very large user-base and growing rapidly. Important to DOE.
- Cray’s optimized Trilinos released on January 21 2010
  - Includes 50+ trilinos packages
  - Optimized via CASK
  - Any code that uses Epetra objects can access the optimizations
- Usage : module load trilinos

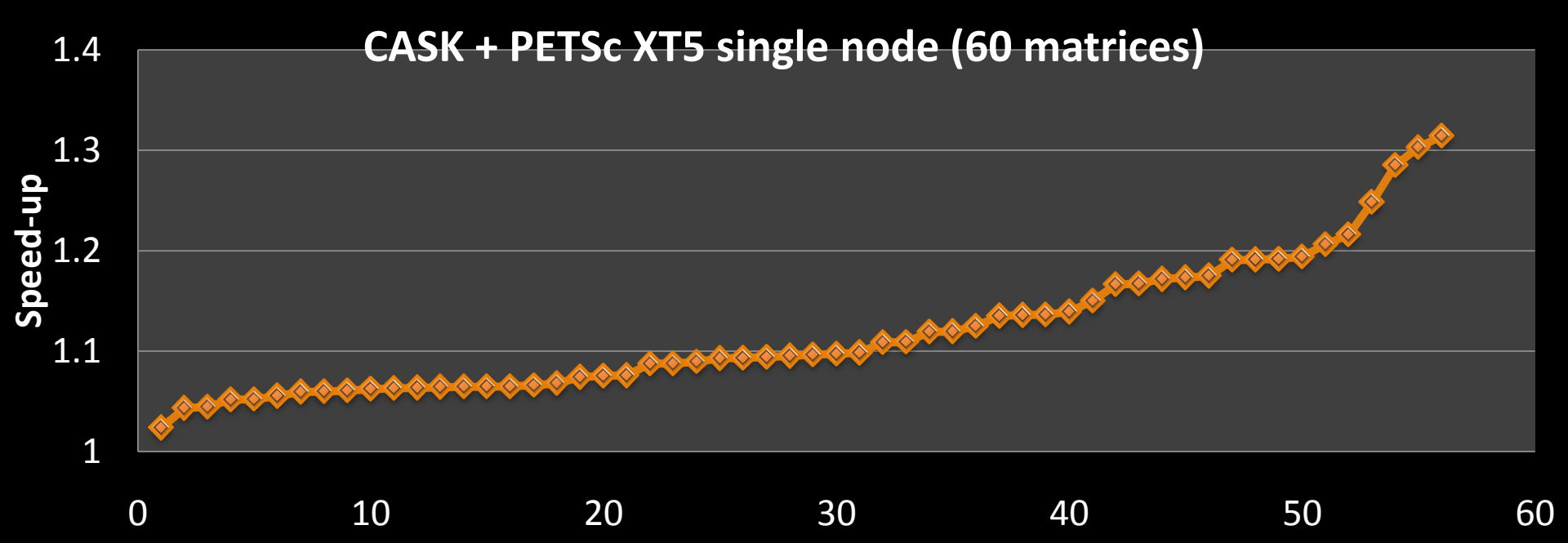
Cray Adaptive Sparse Kernel (CASK)

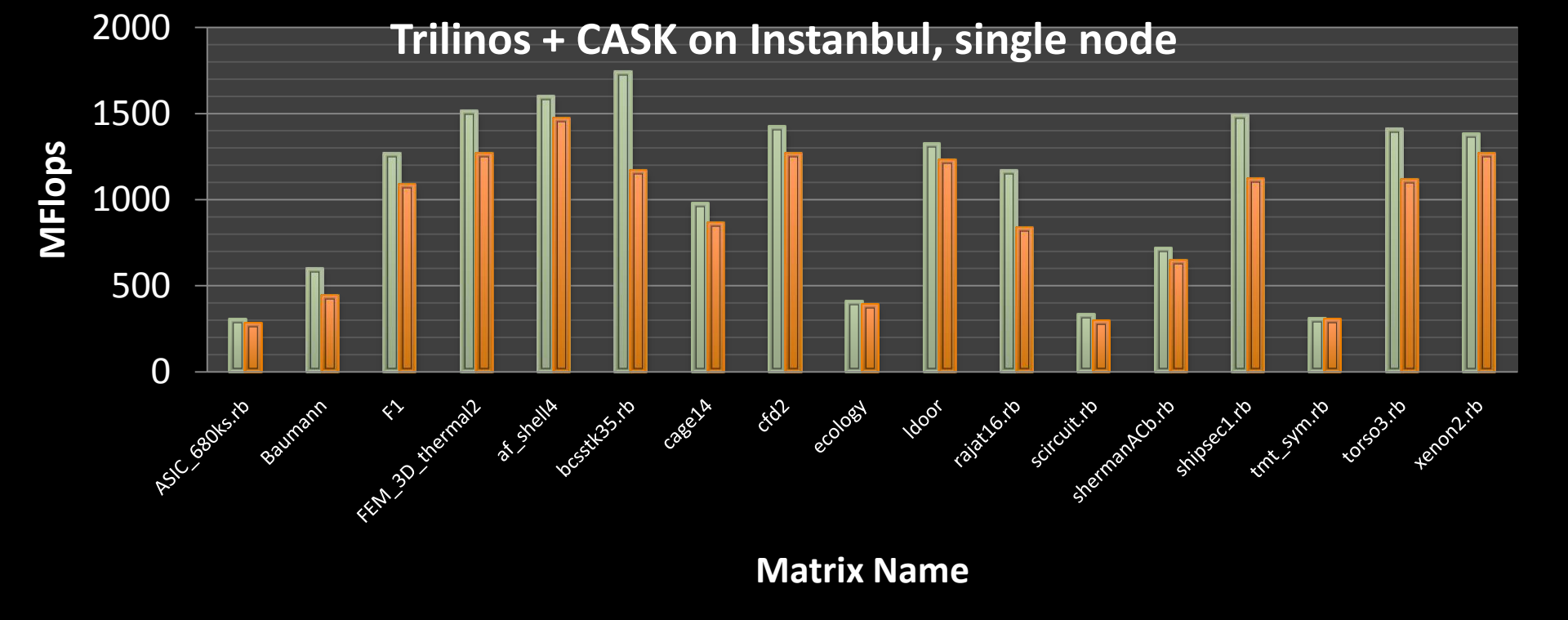
- CASK is a product developed at Cray using the Cray Auto-tuning Framework (Cray ATF)
- Uses ATF auto-tuning, specialization and Adaptation concepts
- Offline :
  - ATF program builds many thousands of sparse kernel
  - Testing program defines matrix categories based on density, dimension etc
  - Each kernel variant is tested against each matrix class
  - Performance table is built and adaptive library constructed
- Runtime
  - Scan matrix at very low cost
  - Map user's calling sequence to nearest table match
  - Assign best kernel to the calling sequence
  - Optimized kernel used in iterative solver execution

Support Model




PETSc Strong Scalability on Shanghai XT5
MFlops
70000
60000
50000
40000
30000
20000
10000
0
PETSc Weak Scalability on Shanghai XT5
N=65,536-16,777,216
MFlops
80000
60000
40000
20000
0
0
50
100
150
200
250
300
# of PEs
PETSc-3.1
PETSC-3.1 CASK


CASK + PETSc XT5 single node (60 matrices)
Speed-up
1.4
1.3
1.2
1.1
1
0
10
20
30
40
50
60


Trilinos + CASK on Instanbul, single node
MFlops
2000
1500
1000
500
0
ASIC_680ks.rb
Baumann
F1
FEM_3D_thermal2
af_shell4
bcsstk35.rb
cage14
cfd2
ecology
ldoor
rajat16.rb
scircuit.rb
shermanACb.rb
shipsec1.rb
tmt_sym.rb
torso3.rb
xenon2.rb
Matrix Name

CASK on MC12 and XE6

- MC12 is the first entirely automated CASK
- ATF used for all stages
  - Codegen
  - Testing, search
  - Execution
  - Automation of adaptive lirbrary
- Released September 2010

CASE – Cray Adaptive Simplified Eigensolver

- Eigensolvers are extremely complicated to use
- Often require quite complicated calling sequences
- Also often require complicated work array set-up
- CASE is a simplified interface into the existing eigensolers
- CASE is also an adaptive framework to use a faster eigensolver

CASE details

- real and complex, serial and parallel wrappers for eigensolvers
- Very simple overloaded/generic interfaces
  - Use a fortran module (‘use case’ in fortran file)
  - Use a C++ header (c users)
- Creates all work arrays for you
- Deduces form the arguments that you pass what type of functionality you require, and calls the best eigensolver for the problem you want
- Can also get adaptive eigensolver by setting CASE_USE_FASTEST
- Now has generic interfacing for both Fortan and C (if using CC)

LibSci_acc - Cray scientific Library for Accelerators

- GPU and hybrid library execution
- BLAS, LAPACK, FFT, Sparse MV
- BLAS is tuned via the auto-tuning framework
- LAPACK is tuned to avoid as much of the communications cost as possible
- FFT is tuned assuming that the
- If you want to obtain accelerated library codes, add –lsci_acc to the link (likely a xtpe-accel module will be available), then relink.

The future of LibSci

- Work with the code developers to make applications scale to the next level
- Prepared to go outside of the bounds of what library vendors normally provide
  - Specialization model, and auto-specialization with training runs
  - Kernel auto-tuning using a framework for advanced users
- Highly optimized hyrbid libraries for CPU and Accelerator


CRAY
THE SUPERCOMPUTER COMPANY